UCLL HOGESCHOOL	Number of pages: 6	Lecturer: J. Pieck, B. Van Impe course unit (OPO/OLA): Back-End Development Code OPO/OLA: MBI03j The **standard time** for this exam is **2h20min**, but each student is given 30% extra time, so **3 hours total**
Date: 05/09/2025 Group: **1BCS**		Student name: Student number: Time of submission: .....hours.....min.

Allowed tools

- The exam is done on your own laptop.
- No smartphone, nor smartwatch.
- Documentation you may use on your laptop:
  - Learning materials we made available on Toledo
  - Pre-made summaries
  - Code your own project and exercises
- Other documentation/tools you may *not* use: internet applications, messaging apps, AI tools (both online and offline)

Composition of the exam

The exam consists of two parts:

- A theoretical part as an ANS test (at 4 points).
- A practical part as an ANS test (on 12 points) including:
  - This assignment as a PDF
  - A zip with additional code
- Access code ANS is projected during the exam.

Startup

- Connect to the wifi network 'ans'. The password is 'examen001'.
- **!! Important:** make sure you are logged out of Postman before the exam. Otherwise, you will not be able to use Postman during the exam.
- Download the zip with additional code from ANS. This contains a first version of the domain classes and schema.sql from which you can start. Place these in the appropriate package of your start code.

Submitting

- You submit your code in one zip file named "LastName_FirstName.zip"
- Submit by uploading this zip to the ANS test.
- Once everything is uploaded, check again in the test to make sure your zip is there. Download your submission to another folder, unzip, and check that you submitted the correct one.
- Hand over this assignment bundle and sign the attendance sheet.

Evaluation

- You can score points for each (partial) question.
- Some (partial) questions are related to others, some are not.
- The functionality must work. Otherwise, you will not get points for that (sub)question.
- End-points must be implemented exactly as described in the API table. Otherwise, you will not receive points for that (sub)question.
- If the functionality works, the code must be implemented as was taught in class. Otherwise you can lose points for a (sub)question.
- You do not need to write tests unless explicitly requested.
- Work in small steps, do not do more for a question than is explicitly asked.

Practical Exam

We are developing a reservation system for conference rooms. Employees of an organization should be able to use this API to look up conference rooms, manage the equipment available, and book rooms for meetings or other gatherings. The system must maintain information about available rooms, their capacity, standard equipment available, and bookings made. In this exam, we will focus on the basic functionality for retrieving information about bookings and managing the equipment belonging to a room.

You will no longer need the User endpoints for this exam. The startup code you receive for the exam is incomplete and you will need to extend it.

Question 1 (1 point)

For this first question, implement an end-point that retrieves all Booking objects (part a). If the optional request parameter "attendees" is included, then only future Bookings (start date after today) are retrieved, where the number of attendees is greater than the parameter (part b).

- Code according to the principles of layered architecture and use JPA.
- Add test data with DbInitializer. Ensure that multiple Bookings with different numbers of attendees are created.

a) Retrieve all Booking objects.

Request Type	GET
URL	http://localhost:8080/bookings
Path Variables	None
Request params	None
Request Body	None
Response code	200

Output success	<pre>[
	    {
	        "id": 1,
	        "employee": "John",
	        "startDate": "2025-05-20",
	        "endDate": "2025-05-22",
	        "purpose": "Project Kick-off",
	        "attendees": 4
	    },
	    {
	        "id": 2,
	        "employee": "Jane",
	        "startDate": "2025-05-24",
	        "endDate": "2025-05-25",
	        "purpose": "Demo",
	        "attendees": 6
	    }
	]</pre>

Continuation of question 1 on the next page.

b) If the optional request parameter “attendees” is provided, then only future Bookings (start date after today) are retrieved, of which the number of participants is greater than “attendees”.

Request Type	GET
URL	http://localhost:8080/bookings?attendees
Path Variables	None
Request params	attendees: int
Request Body	None
Response code	200

Output success	<pre>[
	    {
	        "id": 2,
	        "employee": "Jane",
	        "startDate": "2025-05-24",
	        "endDate": "2025-05-25",
	        "purpose": "Demo",
	        "attendees": 6
	    }
	]</pre>

Question 2 (3 points)

Implement an end-point that allows you to add equipment (Equipment) into the system.
The answer of this end-point is the newly added Equipment item.

a) Implement the addition of Equipment. You will need to extend the domain with a “shared” field.

Request Type	POST
URL	http://localhost:8080/equipment
Path Variables	None
Request params	None

Request Body	<pre>{
	    "name": "Projector Ultra Max Gold (premium edition)",
	    "Notes": "Be careful, very expensive.",
	    "shared": false
	}</pre>

Response code	200

Output success	<pre>{
	    "id": 6,
	    "name": "Projector Ultra Max Gold (premium edition)",
	    "Notes": "Be careful, very expensive.",
	    "shared": false
	}</pre>

Continuation of question 2 on the next page.

b) The name of the Equipment item must be a minimum of 3 and a maximum of 40 characters long. The length of the “notes” field may not be longer than 50 characters.

Request Type	POST
URL	http://localhost:8080/equipment
Path Variables	None
Request params	None

Request Body	{
	"name": "Pr",
	"notes": "This note is way too long and more than fifty characters",
	"shared": false
	}

Response code	400

Output error	{
	"notes": "Notes cannot exceed 50 characters.",
	"name": "Equipment name must be between 3 and 40 characters."
	}

c) The name of an Equipment item must be unique.

Request Type	POST
URL	http://localhost:8080/equipment
Path Variables	None
Request params	None

Request Body	{
	"name": "Projector Ultra Max Gold (premium edition)",
	"Notes": "Be careful, very expensive,
	"shared": false
	}

Response code	400

Output error	{
	"error": "Equipment with this name already exists."
	}

Question 3 (4 points).

For this question, you implement an end-point to add Equipment to a Room (part a). You need to validate on existing Equipment and Room (part b) and ensure that Equipment can only be added to the Room once (question c). In the last sub-question (d) you make sure that Equipment that cannot be shared can only be assigned once to a Room.

a) Implement the happy path where you add Equipment to a Room. A Room has multiple Equipment items and an Equipment item can belong to multiple Rooms. No validation needs to be done for now. The response from this end-point is the Room with the list of Equipment items. Through DbInitializer, make sure you have multiple Rooms and Equipments that can be linked together.

Request Type	PUT
URL	http://localhost:8080/rooms/{roomId}/equipment/{equipmentId}
Path Variables	roomId, equipmentId
Request params	None
Request Body	None
Response code	200

Output success	{
	"id": 1,
	"name": "A1",
	"capacity": 10,
	"equipmentList": [
	{
	"id": 1,
	"name": "Projector",
	"shared": false,
	"notes": "Standard beamer, HDMI/USB-C connection."
	}
	]
	}

b) When you try to assign Equipment to a non-existent Room or non-existent Equipment to a Room, a RuntimeException should be thrown.

Request Type	PUT
URL	http://localhost:8080/rooms/{roomId}/equipment/{equipmentId}
Path Variables	roomId, equipmentId
Request params	None
Request Body	None
Response code	400

Output error	{
	"error": "Room with id 99 not found."
	}
	{
	"error": "Equipment with id 99 not found."
	}

Continuation of question 3 on the next page.

c) If you try to add Equipment to a Room that already contains it, a RuntimeException should be thrown.

Request Type	PUT
URL	http://localhost:8080/rooms/{roomId}/equipment/{equipmentId}
Path Variables	roomId, equipmentId
Request params	None
Request Body	None
Response code	400
Output error	`{ "error": "Equipment already assigned to this room." }`

d) Some Equipment cannot be shared across different Rooms. The "shared" property of Equipment is used for this purpose. If you want to assign Equipment that cannot be shared and is already assigned to one Room to another Room, a RuntimeException must be thrown.

Request Type	PUT
URL	http://localhost:8080/rooms/{roomId}/equipment/{equipmentId}
Path Variables	roomId, equipmentId
Request params	None
Request Body	None
Response code	400
Output error	`{ "error": "Equipment cannot be shared across multiple rooms." }`

Question 4 (2 points)

a) Write the unit tests below for domain class Room. Place the tests in the correct package.

- `givenRoomAndEquipment_whenAddEquipment_thenEquipmentIsAdded`
- `givenRoomAWithNonSharableEquipmentAdded_whenAddSameEquipmentToRoomB_thenRuntimeExceptionIsThrown`

b) Write the unit tests below for the EquipmentService service class. Use mocks here.

- `givenNewEquipment_whenRegisterEquipment_thenEquipmentIsSaved`
- `givenExistingEquipment_whenRegisterEquipment_thenRuntimeExceptionIsThrown`

Question 5 (2 points)

Write an integration test for question 1b: requesting future Bookings with a minimum number of participants.